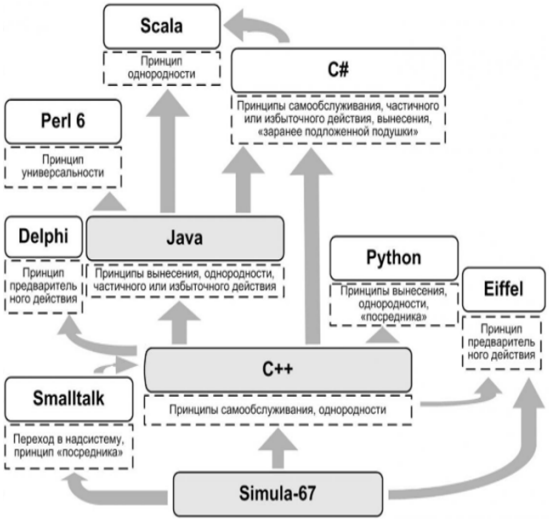

Рисунок – ТРИЗ-эволюционная карта объектно-ориентированных языков программирования

Данная схема позволяет студентам систематизировать знания в области ООП и определить тенденции в развитии языков программирования. Каждый этап представлен в виде наследования: базовый язык формирует последующие этапы.

УДК 004.421.2

Корзун Д. А.

ИСПОЛЬЗОВАНИЕ ЦИКЛОВ И РЕКУРСИИ ДЛЯ РЕШЕНИЯ ЗАДАЧ

БНТУ, г. Минск
Научный руководитель: ст. преподаватель Астапчик Н. И.

Фактически во всех языках программирования циклические структуры можно создавать различными способами. Все они различаются между собой сложностью реализации и требуемыми ресурсами. Рассмотрим и сравним между собой два самых распространенных способа создания циклических структур, а имен-

но используя циклы или рекурсию, благодаря которой можно сымитировать цикл. За основу возьмем язык программирования C++.

Для решения многих задач, в программировании используются циклы. Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Всего в языке программирования C++ 3 вида циклов (for, while, do while). Все они различаются между собой синтаксисом и некоторыми иными характеристиками.

Теперь, рассмотрим, что такое рекурсия. Рекурсия – вызов функции (процедуры) из неё же самой, непосредственно (*простая рекурсия*) или через другие функции (*сложная* или *косвенная рекурсия*). Рекурсию можно использовать для имитации работы цикла, но необходимо учитывать некоторые особенности её работы. В отличие от цикла for, блок кода, который нам нужно выполнить n раз, находится в отдельной функции. Вот именно с этим связаны некоторые сложности. Мы знаем, что все запущенные функции в языке C++ загружаются в стек и не выгружаются пока они не выполнятся. Получается, если функция будет вызывать саму себя, то она не может быть завершена и у нас быстро закончится память в стеке в следствии чего программа утратит работоспособность. Поэтому, нужно добавить условие, при котором и будет происходить эта самая выгрузка. Для этого используется оператор if.

Сравним между собой два метода создания или, как в случае с рекурсией, имитацией циклических структур по производительности кодов. Составим алгоритм, который будет решать головоломку «Ханойские башни» для 16 колец. По формуле $k = 2^x - 1$, где n-кол-во колец, рассчитаем необходимое кол-во необходимых действий k. K=65535 действий. Сравнительный анализ алгоритмов приведен в таблице 1.

Следуя из этой таблицы, можно сделать вывод, что код, использующий цикл, работает на ~13% быстрее, нежели, когда мы используем рекурсию, но при этом он занимает больше оперативной памяти и дольше длится общее время ЦП.

Таблица 1 – Сравнительный анализ потребления системных ресурсов

	Занимаемая память	Общее время ЦП (мс)	Время выполнения (мс)
Рекурсия	8,2 Мб	4216	19897
	8,2 Мб	4340	19792
	8,2 Мб	4291	19823
Итог	8,2 Мб	4282.33	19837.33
Цикл	8,4 Мб	4524	17612
	8,4 Мб	4504	16925
	8,4 Мб	4513	17301
Итог	8,4 Мб	4513.66	17279.33

К плюсам же решения алгоритма «Ханойские башни» с помощью рекурсии, можно отнести простоту написания кода и меньшее количество занимаемой оперативной памяти.

Итак, подведем итог. Вызов функции влечет за собой некоторые дополнительные накладные расходы, связанные с передачей управления и аргументов в функцию, а также возвратом вычисленного значения. Чаще всего итерационные решения работают быстрее рекурсивных.

Еще одним недостатком рекурсии является то, что ей может не хватать для работы стека. При каждом рекурсивном вызове в стеке сохраняется адрес возврата и передаваемые аргументы. Если рекурсивных вызовов слишком много, отведенный объем стека может быть превышен.

Однако процедуры, вызывающие себя два и более раз чаще всего не имеют простого нерекурсивного аналога. В этом случае множество вызываемых процедур образует не цепочку, а целое дерево. Существуют широкие классы задач, когда вычислительный процесс должен быть организован именно таким образом. Как раз для них рекурсия будет наиболее простым и естественным способом решения. Кроме того, рекурсивные алгоритмы, как правило, намного проще с логической точки зрения, чем итерационные.

Код программы с использованием цикла:

```
#include <iostream>
#include <vector>
```

```
using namespace std;
struct State
{     int n;
      int src;
      int dest;
      int tmp;
      int step;};
void tower(int n, int src, int dest, int tmp)
{ vector<State> stack;
      { State state;
      state.n = n;
      state.src = src;
      state.dest = dest;
      state.tmp = tmp;
      state.step = 0;
      stack.push_back(state); }
      {while (stack.size() > 0)
            State &state = stack.back();
            switch (state.step)
            { case 0:
                  if (state.n == 0)
                        stack.pop_back();
                  else
                  {++state.step;
                  State newState;
                  newState.n = state.n - 1;
                  newState.src = state.src;
                  newState.dest = state.tmp;
                  newState.tmp = state.dest;
                  newState.step = 0;
                  stack.push_back(newState); }
                  break;
            case 1:
                  cout << state.src << "->" << state.dest << endl;
                  ++state.step;
```

```
                    State newState;
                    newState.n = state.n - 1;
                    newState.src = state.tmp;
                    newState.dest = state.dest;
                    newState.tmp = state.src;
                    newState.step = 0;
                    stack.push_back(newState);
                    break;
            case 2:
                    stack.pop_back();
                    break; }}}
int main()
{ tower(5, 1, 2, 3); }
```

Код программы с использованием рекурсии:

```
#include <iostream>
#include<ctime>
using namespace std;
void hanoi(int num, char from, char to, char tmp)
{
    if (num != 0)
    {
        hanoi(num - 1, from, tmp, to);
        cout << from << " -> " << to << endl;
        hanoi(num - 1, tmp, to, from);
    }
}
int main()
{
    hanoi(16, '1', '2', '3');
    unsigned int end_time = clock();
    cout << clock() << endl;
    system("Pause");
    return 0;
}
```